Performance Analysis of Different Optimization Methods for Linear Regression

FYS-STK3155 - Project 1

Alexander Umansky
Norwegian University of Science and Technology, University of Oslo
(Dated: October 5, 2025)

In this study, we used different optimization methods to find a polynomial model for the Runge function. We compared standard algebraic approaches, like Ordinary Least Squares, and Gradient Descend methods. Performance was evaluated based on mean squared error, stability of model coefficients, and convergence rate. Adding regularization to moderately stochastic data proved to stabilize the coefficient values and make the model more resilient to overfitting at higher complexities. The optimal model complexity was estimated using prediction-bias and variance of a small test probe, which was re-sampled using Bootstrap and K-fold methods. The optimal model degree is $P = 10$, but with regularization it can be set up to 15. Performing Gradient Descend with adaptive learning rate ADAM, made it possible to replicate the Runge slope without oscillations up to the edges and without seriously collapsing the fit near the origin. We investigated benefits of using Mini-Batch and Stochastic Gradient Descend methods, adapted with Importance Sampling of batches. Although convergence did improve, the overall fit worsened due to stochastic noise. Importance sampling prevented loss of precision around the edges, but elsewhere the fit got broadened.

I. INTRODUCTION

The optimization problem is central to any field of research, where we seek to match observations with data generated by models. Intuitively, the closer the alignment between the true and synthetic data, the more likely the model is to correctly represent reality. However, how should "matching" be measured, and wether good alignment always signifies correctness of the model, are central topics of Machine Learning and this study in particular.

This study is limited to Linear regression where features are polynomial degrees x^n up to some maximal model complexity $n \leq P$ and the optimization problem is to find polynomial coefficients $\hat{\beta}$ such that $y \simeq \hat{\beta}X$. Though we can assume that the parameters are independent of the input data, models of high complexity can be computationally demanding, hence we investigate performance of algebraic, Ordinary Least Squares (OLS), and Gradient Descend (GD) based approaches. Performance will be based on the mean squared error (MSE), stability and consistency of coefficient values, and for GD methods, also the convergence. We will investigate how adding regularization λ to OLS method affects the performance, and how updating the learning rate η improves parameter fitting for non-trivial functions.

The final topic of the analysis concerns how limited datasets can be used more effectively to improve optimization and training. In the context of Gradient Descend we look for convergence benefits of Mini-Batching and Stochastic Gradient Descend (SDG) with uniform and importance batch-sampling. We also discuss Bootstrapping and K-fold cross-validation as two resampling methods to regenerate test data. We study the balance between model complexity and amount of training data required for optimal prediction-bias and variance.

II. THEORY AND METHOD

A. The Runge Function

In this work we study the Runge function $y(x) = \frac{1}{1+25x^2}$ on the interval $x \in (-1, 1)$ polluted by normally distributed noise $N(0, \sigma \leq 1)$. The choice of the Runge function is not a coincidence as it is a case where polynomial approximation of increasing order does not converge and instead produces larger oscillations near the edges of the interval[1]. An analogy exits in Fourier analysis, called the Gibbs phenomenon, where approximating non-smooth functions with higher modes produces oscillations that never truly die out. Without limiting the model complexity, one solution involves using orthogonal series, such as Chebyshev polynomials, to control the oscillations [1][2]. In this project, we stay true to polynomial regression, in order to stress test the different methods.

B. Preprocessing and Evaluation

Once the $(N \times P)$ feature matrix X is constructed, where N is number of data points and P is model complexity (i.e. highest polynomial degree), we standardize each feature column to have zero mean and unit variance. This is done to ensure that each feature initially has equal weighting in the analysis. Another reason is that ill-conditioned feature matrices will accumulate floating-point errors during repeated matrix operations [3]. In later sections where we introduce regularization, feature scaling will become even more important in context of penalization.

The data is split into training and test subsets using a 1:4 ratio. Model parameters are optimized for the training data, while the test data is used to evaluate how good the model is at generalization by predicting unseen data. Before training, we scale using a 'standard scaler' pro-

vided by scikit-learn library to scale X_{test} and X_{train}. We subtract the mean value from y_{test}, which will be added later during evaluation as an offset. This prevents scaling from introducing bias into the optimization.

One performance metric will be the test error. Here we define the Mean Squared Error and R^2:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2 \tag{1}$$

$$R^2 = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2} \tag{2}$$

where $\tilde{y} = X_{test}\hat{\beta}$, and $\bar{y}$ is the mean of test data y. The attractiveness of MSE is first of all due to its simplicity, and how it seamlessly emerges in so many different domains like Ordinary Least Squares in linear algebra or Bias-Variance in statistical analysis, thereby bringing them into the context of optimization.

The R^2 metric normalizes MSE by dividing it by the variance of y. Hence R2 is less dependent on preprocessing and its values are dimensionless.

C. Optimization

We obtain the optimal parameters by minimising the cost function, without regularization λ, defined as ordinary least squares (OLS) or the MSE from before:

$$\mathcal{C}_{OLS}(\beta) = \frac{1}{n}\|X\boldsymbol{\beta} - y\|_2^2 = \frac{1}{n}(y - X\beta)^T(y - X\beta) \tag{3}$$

The optimal parameters are obtained by minimizing the cost function i.e. $\nabla\mathcal{C}_{OLS}(\beta) = 0$. If X and y are non-stochastic and independent of parameters, an algebraic solution for β exists:

$$\hat{\beta}_{OLS} = (X^TX)^{-1}X^Ty \tag{4}$$

Here we define the Hessian matrix $H = X^TX$. The Hessian happens to be the second derivative of the cost function and proportional to the covariance matrix. Its diagonal elements are variances, while its eigenvalues describe the type of optimization problem[1][4][5].

The matrix solution for the optimization problem relies on the existence of a matrix inverse of H which may well not be the case for high complexity and large degree of dependencies between the feature columns. The solution is to use Singular Value Decomposition (f.ex. NumPy's pseudoinverse), or introduce a small regularization λ to the diagonal terms in H. We define the Ridge and Lasso regularization methods:

$$\begin{aligned}\mathcal{C}_{\text{Ridge}}(\beta) &= \frac{1}{n}\|X\beta - y\|_2^2 + \lambda\|\beta\|_2^2 \\ \implies \hat{\beta}_{\text{Ridge}} &= (X^TX + \lambda\mathbf{I})^{-1}X^Ty\end{aligned} \tag{5}$$

$$\mathcal{C}_{Lasso}(\beta) = \frac{1}{n}\|X\beta - y\|_2^2 + \lambda\|\beta\|_1 \tag{6}$$

The second term in equations 6 and 5 is called the penalty term. Regularization parameter does not just solve the issue of invertability, it realizes a size constraint on the parameter coefficients:

$$\hat{\beta}_{\text{Ridge}} = \arg\min_{\beta}\sum_{i=0}^{N} -1\left(y_i - \beta_0 - \sum_{j=1}^{P-1} x_{ij}\beta_j\right)^2 \quad \text{subject to } \sum_{j=1}^{P-1}\beta_j^2 \leq t. \tag{7}$$

This expression simply rewrites and merges the L2 norms of Eq 5, so the t is related to λ [5]. Written this way, one can see that neglecting feature scaling results in uneven penalty between jth features. If before, scaling X with some diagonal matrix D: ($D^T = D$), simply scaled the coefficients $\hat{\beta} = (DDX^TX)^{-}DX^Ty$, with regularization this is no longer the case. Notice also $y_i - \beta_0$. By subtracting the intercept $\simeq \bar{y}$ we make the penalty independent of data offsets (value size)[5].

When model parameters are highly correlated (such is the case for polynomial regression), regularization is a way to enforce stability of coefficients, the second performance milestone (somewhat argued in [5]). Finally, since the Lasso method has a L1 penalty term, the solution is no longer linear in y as the gradient will contain the term $sgn(\beta)$ in Eq. 10. Hence, no algebraic solution exists for the Lasso regularization unlike Ridge or OLS.

D. Statistical Interpretation

Consider $y = f(x) + \varepsilon$ where ε is some noise parameter given by $N(\mu = 0, \sigma)$ and f is assumed non-stochastic, independent on distributions of y and X. Then, the expected values of the model prediction $\tilde{y} = X\hat{\beta} \simeq f(x)$ and the true value y, are both given by $X\hat{\beta}$. If we want to express the expected value of their squared differences as in Eq. 1 it can be shown [4] that $\mathbb{E}[(y - \tilde{y})^2]$ relates MSE to bias and variance of the prediction sample i.e. the distance from the average prediction to the true value, and the dispersion of predictions.

$$\mathbb{E}\big[(y - \tilde{y})^2\big] = \underbrace{\mathbb{E}\big[(y - \mathbb{E}[\tilde{y}])^2\big]}_{\text{Bias}} + \underbrace{\mathbb{E}(\tilde{y} - \mathbb{E}[\tilde{y}])^2\big]}_{\text{Variance}} + \underbrace{\sigma^2}_{\varepsilon \text{ noise}} \tag{8}$$

[1] If the Hessian is semi positive definite, meaning its eigenvalues are $\lambda_H \geq 0$, this implies a convex optimization problem i.e. finding the local minima of the cost function $\mathcal{C}(\beta)$.

The benefit of this will become clear when we analyze what model complexity to stop at. For instance, a polynomial of high degree can be fitted to one set of data and have a seemingly good MSE value, yet diverge from the true function in some other region. This model will give good results for a particular test data, but exhibit high prediction variance.

E. Gradient Descend

In this section we introduce Gradient Descend (GD) as a more rigid optimization method. For high model complexities, computing the inverse (with or without SVD) becomes computationally unaffordable. Besides, any optimization problem which is dependent on parameters (f.ex. Lasso) has no analytical expression to take advantage of. GD builds upon the Newton method to find the roots of the cost function iteratively. We can expect fastest convergence when following the negative gradient of the function [4] i.e. $\nabla_\beta \mathcal{C}(\beta)|_{\beta^n} \equiv g^{(n)}$ such that:

$$\beta^{(n+1)} = \beta^{(n)} - \eta g^{(n)} \tag{9}$$

$$g^{(n)} = \frac{2}{n} X^T (X\beta^{(n)} - y) + \begin{cases} 0 & \text{(OLS)} \\ 2\lambda\beta^{(n)} & \text{(Ridge)} \\ \lambda \operatorname{sgn}\left(\beta^{(n)}\right) & \text{(Lasso)} \end{cases} \tag{10}$$

where η is the Learning Rate. For cases where analytic solution still exists, we essentially traded inversion of an inner product of two $N \times P$ matrices $\mathcal{O}(NP^2 + P^3)$, for less demanding iterative computation of the gradient $\mathcal{O}(NP)$ per iteration. Of course, the Gradient Descend has its own shortcomings, for once, re-computation of the gradient can too become inefficient. Besides, the major downside of GDs is that the methods finds only *local* minima and is very dependent on initial conditions.

One practical solution is to subdivide test data into M mini-batches of some size S, and let each approach independently their local minima. The full data set (batch mode) and Stochastic Gradient Descend (batches consist of individual points) are limit cases of Mini-Batching. Each iteration, or epoch (e), we randomly construct M batches of size S (instead of choosing 1 fixed batch at random[4]). The global update for one epoch becomes:

$$\beta^{(e+1)} = \beta^{(e)} - \eta \sum_{m}^{M} \sum_{i \in B_m^{(e)}}^{S} \nabla_\beta \mathcal{C}_i(x_i, \beta) \tag{11}$$

Another downside is that eq.9 is "isotropic" in parameter-space [4]. We could include higher order expansion, involving the Hessian, but this is again a computational issue. Instead we opt to "modify" the learning rate η. If set too small or too high, GD will fail to converge efficiently. Hence, η must reflect the smoothness of the direction i.e. be a function of the gradient history. In this report we consider adaptive learning rate methods: Momentum, AdaGrad, RMSProp and ADAM from [6].

1. Momentum

Similar to the standard approach behind Verlet algorithms, one can introduce memory by considering current and previous iterations when computing the update[7]:

$$\begin{aligned} \beta^{(n+1)} &= \beta^{(n)} - (\eta - \delta) g^{(n)} - \delta\left[\beta^{(n)} - \beta^{(n-1)}\right] \Leftrightarrow \\ v^{(n)} &= \rho v^{(n-1)} + (1-\rho) g^{(n)} \\ \beta^{(n+1)} &= \beta^{(n)} - \eta v^{(n)} \end{aligned} \tag{12}$$

2. Adaptive Gradient (AdaGrad)

Adaptive Gradient method accumulates squared gradients, and uses it to shrink the learning rate:

$$\begin{aligned} r_j^{(n)} &= r_j^{(n-1)} + g_j^{(n)} g_j^{(n)} \\ \beta_j^{(n+1)} &= \beta_j^{(n)} - \frac{\eta}{\sqrt{r_{n,j}} + \epsilon} g_{n,j} \end{aligned} \tag{13}$$

here ϵ prevents division by zero, and double indexing $g_j^{(n)}$ denotes accumulative value of the gradients.

3. RMS Propagation (RMSProp)

A more gradual learning rate scaling than AdaGrad, uses the step averaging from the momentum scheme, but with squared gradient:

$$\begin{aligned} r^{(n)} &= \rho r^{(n-1)} + (1-\rho)(g^{(n)})^2 \\ \beta^{(n+1)} &= \beta^{(n)} - \frac{\eta}{\sqrt{r_n} + \epsilon} g^{(n)} \end{aligned} \tag{14}$$

4. Adaptive Momentum (ADAM)

Adam method combines both moments of RMSProp and Momentum (using $n\rho_i$ as $\rho_i^{(n)}$ for bias correction [4]):

$$\begin{aligned} r_1^{(n)} &= \rho_1 r_1^{(n-1)} + (1-\rho_1) g^{(n)}, \quad \hat{r}_1^{(n)} = \frac{r_1^{(n)}}{\sqrt{1 - n\rho_1}} \\ r_2^{(n)} &= \rho_2 r_2^{(n-1)} + (1-\rho_2)(g^{(n)})^2, \quad \hat{r}_2^{(n)} = \frac{r_2^{(n)}}{\sqrt{1 - n\rho_2}} \\ \beta^{(n+1)} &= \beta^{(n)} - \eta \frac{\hat{r_1}^{(n)}}{\sqrt{\hat{r}_2^{(n)}} + \epsilon} \end{aligned} \tag{15}$$

F. Bootstrap and K-fold Cross Validation

Resampling techniques are used to regenerate new test data, which is particularly important when the dataset is sparse. In bootstrapping, the training sample is repeatedly resampled with replacement. This allows duplicate values to occur, but preserves the distribution of X_{test}. K-Fold Cross validation partitions the X sample into K "folds". For K iterations, a new fold serves as test data while the other are used for training. The predictions are recalculated K times, and then averaged.

In this report, we apply resampling to evaluate model complexity by examining how bias, variance and MSE change as function of P and N. Since N must remain relatively small to deduce optimal P, we generate additional data using $B > N$ bootstraps. We run a larger analysis, comparing OLS to Ridge, using K-Fold cross validation with $K \in (5, 20)$.

G. Resources

For the simulations, we wanted to rely as much as possible on our own code, ensuring full control over the methods, scaling and resampling techniques. The same preprocessing scheme was used across all comparisons, so that any observed differences can be attributed solely to the parameter we changed on purpose. Library usage:

- **Scikit-learn**: StandardScaler, `train_test_split`, KFold and resample.
- **NumPy**: linalg.norm, pseudoinverse, lin. algebra.
- **Matplotlib**: plot generation.

We also made active use of OpenAI: ChatGPT and DeepSeek to identify bugs and logical inconsistencies in both the code and our comparative approach.

The code with necessary packages can be accessed here: https://github.com/4Lexium/Data-Analysis-and-Machine-Learning/tree/main/project1

III. RESULTS AND DISCUSSION

A. Preprocessing and Ordinary Least Squares

In section II A we did not specify the variance of the noise component. We have tried different parametrization of noise and found $\sigma = 0.3$ to be harsh enough for our trials. Figure 1 demonstrates such noisy distribution. The yellow line highlights the true bell-shaped slope. The largest concern is overfitting near the edges. There, the model will use higher polynomials to fit points that are displaced due to noise. Since the error evaluation using MSE uses test data with the same problem, it will not detect this as an issue. Using the target function would be "a posteriori", undesirable for generalization.

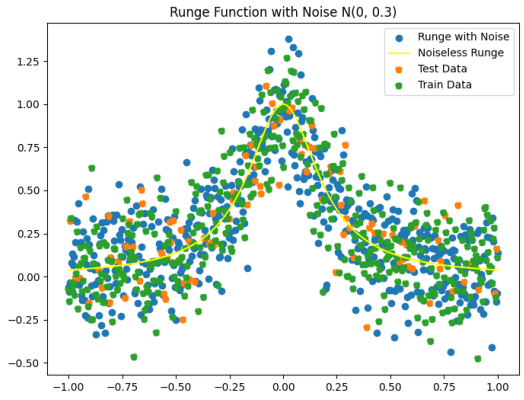

Figure 1: The preprocessed noisy ($\sigma = 0.3$) data sample (blue) is split into train and test subsets (green and orange) and plotted along with the noiseless Runge function (yellow). The y_{offset} is intentionally left out during optimization and added back during prediction evaluation.

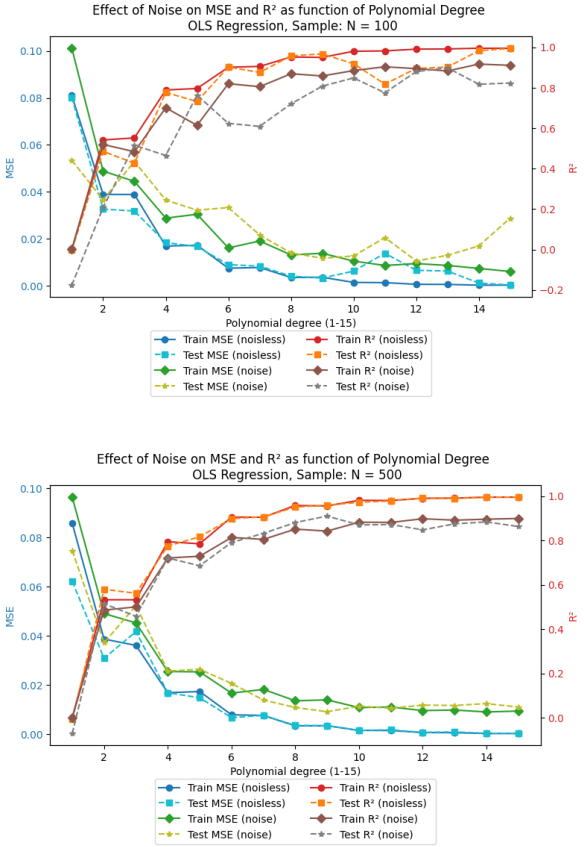

Figure 2: Error estimates using MSE and R2 are used here to compare performance of OLS method on a noiseless sample and one polluted by $\mathrm{N}(\mu = 0, \sigma = 0.1)$. Despite the evident worsening when introducing noise, when large samples are used $N = 500$, the error fluctuations are controlled.

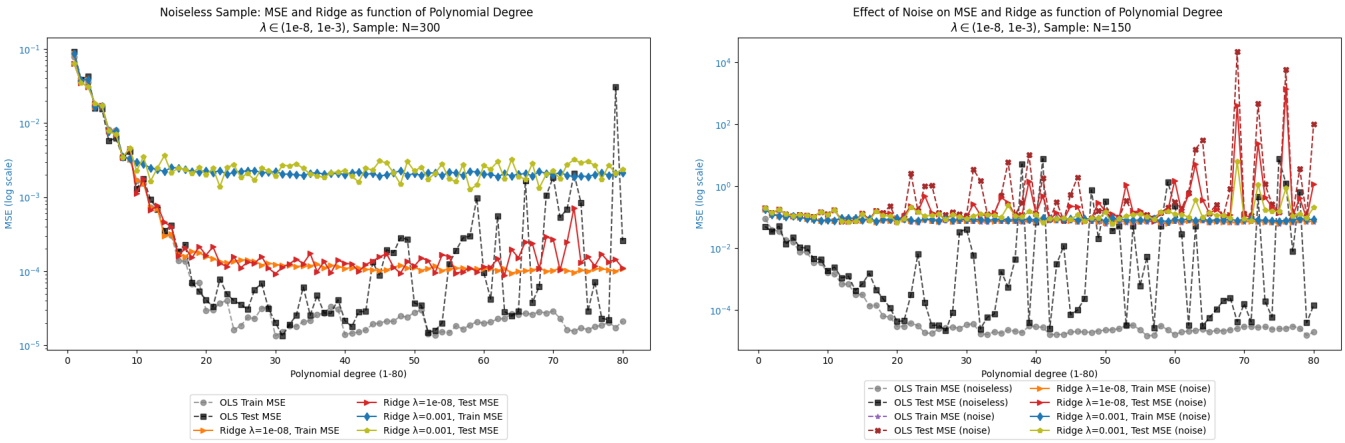

Figure 3: Comparison by log MSE demonstrate how Ridge and OLS methods perform on different datasets sizes without/with noise ($\sigma = 0.3$). In noiseless case, regularization provides stable coefficients and suppresses error fluctuations. Ridge is more advantageous for smaller noiseless samples. Same sign regularization, fairs badly with stochastic noise, making OLS competitive for noisy samples with larger number of data points. Still, regularization reduces the risk of overfitting at higher model complexities.

Figure 2 shows results of OLS optimization of a data sample with noise $\sigma = 0.1$. The data was preprocessed (section II B) beforehand, so the scale dependent MSE and scale independent R2 agree on performance. The former decreasing to 0 while the latter approaching 1 when more features are added. Notice already after $P = 10$, the improvement starts to stagnate and what began as random jumps in prediction error, resemble beginning of divergence.

At this stage we use a mild level of noise and relatively normal complexity. This test shows in particular how sample size affects OLS. If enough points are used in training, even if globally, noisy prediction still worsens, the error fluctuations are more controlled. There is also no sign of divergence at this concrete complexity interval. This topic will be re-visited in section III C.

B. Introducing Regularization

In this section we study Ridge regression i.e. introduce regularization. We choose to study $\lambda \in (10^{-8}, 10^{2})$ capturing both extreme cases of regularization. If we start with a noiseless sample, as $\lambda \to 0$ the Ridge method converges towards OLS with same MSE error as seen in Figure 3 (left). When λ is at the same magnitude as the entries of $X^T X$, roughly $X_{ij}^2 \sim 10^2$, the penalty term of eq.5 significantly perturbs the original problem. In such a state, optimization of the cost function is not as much fitting target y, as simply scaling down the parameter coefficients. Figure 6 shows a 2D plot $MSE(P, \lambda)$, how error depends on both complexity and the introduced regularization. The just mention issue with extreme regularization is clearly highlighted.

Another noticeable trend is how the MSE colour gradient goes along complexity P, and varies slightly for intermediate regularization values. Applying adequate penalization results in more stable parameter coefficients than without regularization. Penalization also mitigates dependencies that inevitable arise when fitting a polynomial model with large number of parameters. The fact that MSE does not change monotonically, neither keeps improving nor worsening, leads us to believe that adding regularization to optimization reduces the chance of overfitting with larger model complexity, more so than methods without regularization.

Figure 3 (left) further emphasizes the value of stability achieved with regularization at high model complexity, entailing many more dependencies. Indeed, penalization helps mitigate the problem with dependent features and the error fluctuations. However, when significant noise ($\sigma = 0.3$) is added, Figure 3 (right), methods with regularization perform equally worse as OLS. Here, the stabilizing advantage of penalization is surpassed by it perturbing the problem in one direction, while noise is stochastic. Therefore, it is incorrect to judge wether OLS or Ridge is better. Regularization is worse at fitting stochastic data, but OLS has major performance issues for small data samples and is more exposed to risk of overfitting at lower complexities.

C. Prediction Bias and Variance

So far we have only compared performance without any fixed complexity. In this section, we determine the optimal number of parameters by studying a more advantageous metric than MSE, prediction-bias and variance, which were showed to be related measurements via eq.8. We choose smaller data samples to test how the

model generalizes, and apply Bootstrap or K-fold cross validation to resample from the same data distribution.

Figures 4 (top and middle) are to be interpreted the following way. Sudden steepening in variance signifies overfitting, implying that the model complexity is too high. What happens is a highly curving polynomial displays good MSE as it has been tailored to fit a small number of train points, but might trail off everywhere else. Hence, a slightly different test set, and prediction is off. Resampling mitigates the chance of unlucky dataset, and averaged, it will describe a generic prediction performance for given P. On the other hand, low variance and high bias implies the model is consistently too simple. We seek a balance, hence the name tradeoff, which usually lies in the minimum of the resulting curve. To analyze larger complexities the sample number has to be increased, hence we fix P and vary N and/or number of bootstraps, folds. Note therefore, that the optimum is a dependent value, assumed generic for datasets only of similar size as those considered in this study.

From Figure 4 we conclude that OLS model complexity should not be larger than $P = 10$, which is consistent with what was discovered in section III A. Doing same procedure, but for Ridge regression, reveals that the optimum is slightly shifted past $P > 10$. Best phrasing would be that regularization attenuates and smoothens the variance slope, whereby how much depends on the regularization parameter λ. Note that the same sample size and number of bootstraps have been used when directly comparing different regularizations to OLS. This validates our earlier statement that penalization reduces the danger of overfitting.

We run a different resampling algorithm to avoid tuning N and number of bootstraps for each P, instead having to specify only the number of folds. Figure 4 (bottom) uses $K = 10$, but the outcome is no different for 5, 15. The prognosis is consistent in that $P = 10$ is the optimum, that Ridge regularization defers the risk of overfitting, and that fitting with degree lower than the optimum results in large bias.

D. Gradient Descend Method with Constant Learning Rate

The remaining sections will concern Gradient Descent methods, where MSE, parameter coefficients, overall fit and convergence will be the metrics of comparison. In particular we are interested in performance of different adaptive methods and wether Mini-Batching or SGD will yield improvement. An all-encompassing systematic study would indeed be daunting, hence we resort to fixing what has been determined so far, mainly: $\sigma = 0.3, P = 15, N = 500, \lambda = 10^{-5}$ and probe further the best performing methods.

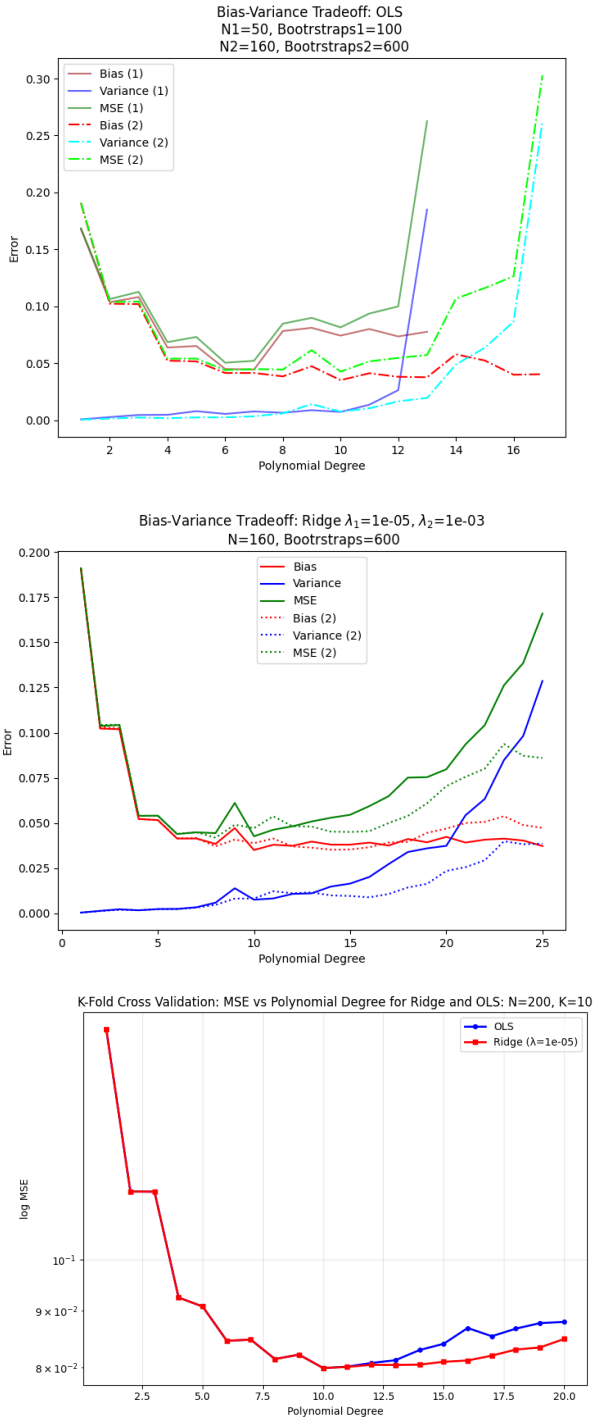

Figure 4: **Top**: Bias-Variance tradeoff for two different train samples with different number of bootstraps. The minimum of each MSE curve determines the optimal complexity. The dashed lines describe results of a larger train set. The optimal complexity indeed depends on N, but a generic estimation can be assumed for test sets that are not too different from the ones used in this study. **Middle**: Same analysis but for Ridge regression with 2 different λ. Higher model complexity without drastic overfitting can be used when applying regularization. **Bottom**: Performance of $K = 10$ folds on a dataset of 200 data points. We observe the same optimal complexity around $P = 10$ for both methods, but Ridge can have more degrees without danger of overfitting. Complexities lower than 7, result in larger bias i.e. too simplistic model.

First, we compare GD with constant learning rate η to the analytic solutions from before. Figure 5 (top) plots the convergence of GD parameters using the OLS and ridge gradient from eq.10. together with the ones obtained from eq. 4, 5. Another consequence of OLS coefficients not being penalized is that nothing prevents a modeling like: $1000x^8 - 2000x^7$, since for $|x| \leq 1$ it contributes nothing to MSE. Were this a model with some physical interpretation, it is evident that such coefficients would be nonsense. Convergence of GD depends on an adequate choice of η. The method will never converge with a small value, but also if $\eta \geq \frac{2}{max(\lambda_H)}$, where λ_H is the largest eigenvalue of the Hessian $H = X^T X$, any gradient based method will diverge[4].

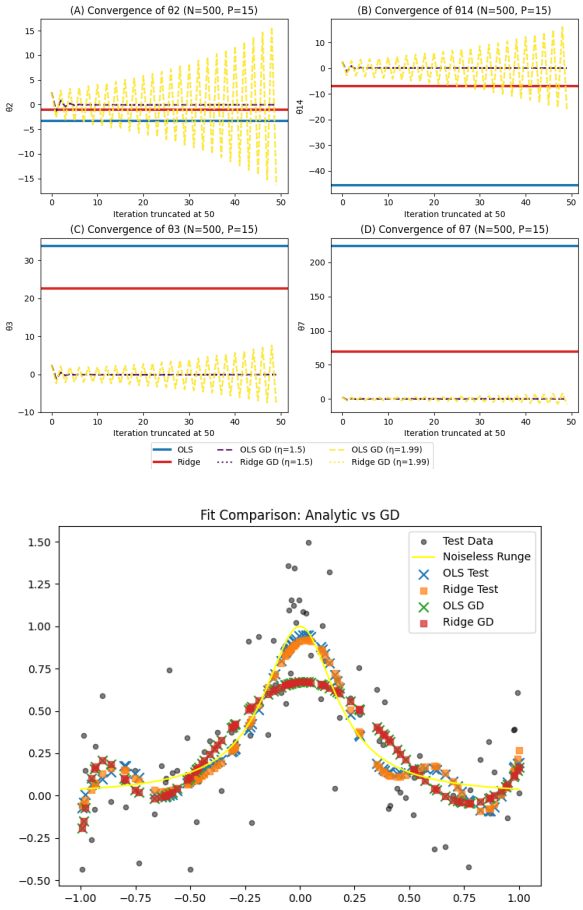

Figure 5: **Top**: GD methods determine the coefficients iteratively. The red and blue lines highlight coefficient found using algebraic expressions (4, 5). Notice how Ridge coefficients are always constrained due to penalty, while OLS coefficients can have very high values. The difference in using OLS or Ridge gradient is insignificant at the given scale. When using constant learning rate, avoid using $\eta = 2/max(\lambda_H)$, where $H = X^T X$ is the Hessian, as any GD based method will diverge. **Bottom** the final fit using the coefficients obtained from analytical coefficients and the final GD iteration. The former describe the curve around the origin much better, but both methods fail to follow the slope near the edges.

For OLS and Ridge GD with $\eta = 1.5/\lambda_H^{max}$ the coefficients converge to values close to zero. Although not necessarily wrong, it is possible we find only a local minima. The parameters were initiated as $\beta^{(0)} = \{2.5\}_p$. Using analytical parameter values for initialization is possible, but this would make our method dependent on the existence of analytical solutions, which we will not assume for the sake of a general argument. The bottom plot compares the fits using analytical and GD based procedures ($\eta = 1.5/\lambda_H^{max}$). The analytic solutions describe the Runge function much better around the origin. Both methods incorrectly track noisy components near the edges, mistakenly adding ripples to the bell shape. This is also a flaw of polynomial approximation of the Runge function, discussed in II A. However, while the analytic expression is set, the GD approach can be adapted with this problem in mind.

E. Adaptive Learning Rate and Lasso Regression

Table below lists the configurations and definitions of parameters used in Eq. 12, 13, 14, 15.

Table 1: Defaults and Test Configurations for Gradient Descend.

Common	$N = 500$, $P = 15$, $\sigma = 0.3$, Initialization: $\beta^0 = \{2.5\}_p$
Defaults	tol $= 10^{-6}$, momentum $= 0.9$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$
Test D	max_iter $= 10^3$ Optimizers: OLS, Vanilla GD, Momentum, Adagrad, RMSProp, Adam Learning rates: [0.055, 0.1, 5, 0.008, 0.55]
Test E	max_iter $= 10^3$ Optimizers: RMSprop, Adam Regularization: Ridge, Lasso, $\lambda = 10^{-5}$ Learning rates: RMSProp $= 0.008$, Adam $= 0.55$
Test F	Adam - Ridge $\lambda = 10^5$ Sampling: Batch, Mini-Batch*, Stochastic, Custom-Sample** Max iterations: [1000, 1000, 2000, 2000] Max Epoch: [-, 100, 200, 200] Learning rates: [0.55, 0.55/2, 0.55/6, 0.55/8] *batch size $= 64$, **Importance Sampling: 0.6σ

Consider Figure 7 where we compare performance of constant learning rate (blue) to that which depends on gradient history, in other words, follows the topography of parameter space. Except for initial learning rate (see Test D), the same configurations were used as previously. With these conditions, adaptive method displays a more precise fit (Plot C) at the cost of slower convergence. Particularities of each method are spesified in detail below.

Momentum is the simplest addition of memory. The derivation uses the same scheme of finite-difference discretization of a 2nd order differential equation as the Verlet algorithms, but expanding in the direction of past iterations $n - k$, and with $m = 1$. Hence v in Eq.12 is an analogy to velocity of a parameter-particle moving in parameter space[6]. Rewriting 12 with older steps yields:

$$v^{(n)} = (1-\rho)g^{(n)} + \rho(1-\rho)g^{(n-1)} + \dots$$
$$v^{(n)} = (1-\rho)\sum_{k=0}^{n} \rho^k\, g^{(n-k)} \tag{16}$$

This shows that learning rate is updated with exponentially decaying gradient average. Eq.16 and equivalently 12 are analogous to that of a digital low-pass filter. The weight ρ is a fixed hyper parameter, but it can also be a function of iteration step like in ADAM: $\rho_i^{(n)} = n\rho_i$. Judging the orange curve in 7C, momentum GD has the best prediction around the origin and worst oscillatory behavior near the edges. In a sense, momentum GD imitates the analytic solutions, also seen in the large coefficient values. To show why, perform a linniarization of the gradient $g = H\beta = \lambda_H\beta$, where eigenvalues incode the curvature of the cost function. Inserting this into 16 and 12 up to $n-1$ iteration we have:

$$\beta^{(n+1)} \approx \beta^{(n)} - \eta\lambda(1-\rho)\beta^{(n)} - \eta\rho\lambda(1-\rho)\beta^{(n-1)}$$
$$D = \big(1-\eta\lambda(1-\rho)\big)^2 - 4\rho\,\eta\lambda(1-\rho) \tag{17}$$

where D is the discriminant of the characteristic equation in β. Oscillations occur for a negative discriminant which is possible for either too large ratio η/ρ or curvature λ. This explains why momentum fits the top of the bell curve with minimal curvature so well.

AdaGrad shrinks the learning rate by the root of *cumulative* squared gradient. The idea is to slow down the updating when large changes are detected. For a convex slope this method is the first to converge (green curve in plot 7A). The Runge function has also a large flat portion, which is why Adagrad is not the method to improve the edge fluctuations.

For a general shape with varying curvatures a cumulative shrinking is rather disadvantageous. RMSPropagation copies the low pass scheme 16 from momentum approach and uses it with gradient squared. This method required the smallest initial η and consequently has the longest convergence rate (still < 500), but the fit follows the bell shape without diverging for a longer interval.

The best fit is achieved by combining the principle of RMSProp and momentum. In addition, Adam introduces bias correction to its moments $\hat{r}_i^{(n)}$ of Eq.15. The bias correction is made iteration-dependent by using hyper parameters $\rho_i^{(n)} = n\rho_i$. This ensures stability at earlier stages[6]. Adam fit (purple) follows the flatter section of the true slope best, while maintaining a good prediction around the origin.

The previous results were obtained without regularization. We now bring in the non-linear Lasso regression and compare the best performing adaptive methods, RMSProp and Adam (see Test E for configurations). Figure 8 shows a qualitatively similar performance as for the case without regularization. A noteworthy difference is that for RMSProp, the choice between $L1$ (Lasso) and $L2$ (Ridge) regularization separates the fits near the edges. This suggests that the weaker gradient signal of $L1$ (without the extra factor 2) is less competitive in this regime. This might indicate that Ridge regression is more preferable for edge-case stability.

F. Batch and Stochastic Gradient Descend

The final topic covers Mini-Batching and Stochastic GD which were motivated in II E. SDG and mini-batch algorithms typically select one random batch per epoch, this offers a clever way to navigate local minima while accelerating convergence. The tradeoff is that these methods can only approximate the true gradient. In this study, we explored a modified approach: reshuffle the batches each epoch, then sum over all batch-contributions, effectively introducing an additional $\sum_m^M$ in Eq.11. Importance sampling was investigated, where batches are drawn from a certain distribution. For our purpose, a Normal distribution is considered to reduce sensitivity to edge noise. For future exploration, using Chebyshev nodes[2][1] is a worthwhile prospect.

Figure 9 shows the results of Test F configurations. Plot A demonstrates how convergence rate improved for SDG and mini batching, despite the fact that learning rate was decreased for sampling methods to compensate for the increased noise in updates. SDG has the fastest convergence, but finds a very different set of parameters than batch and mini-batch. Judging by the overall fit in C, all sampling methods broaden the batch shape, especially the SDG with importance sampling. This method seems to fit well around the edges, but performs worse elsewhere. The mini-batch fit is most consistent with that of batch-mode, which is not surprising considering batch GD is the noise-free limit of mini-batching.

IV. CONCLUSION

In this work, we systematically compared different methods for optimization and several modifications for both analytic expressions and gradient descend approaches. The Runge function with a stochastic component constituted a challenge for predicting the function near the edges. In these regions, the target function has a shallow slope, which high order polynomials can't replicate due to the noise, resulting in oscillations that build up towards the edges. It was therefore important to first figure out the optimal model complexity, and then compare how different optimization methods, with now fixed number of features, trace the shape of the Runge function over the full interval.

For Ordinary Least Squares, increasing complexity beyond $P = 10$ led to increasing prediction variance, signaling overfitting. Modeling with complexities lower than 7 will leed to large bias, and oversimplified models ...

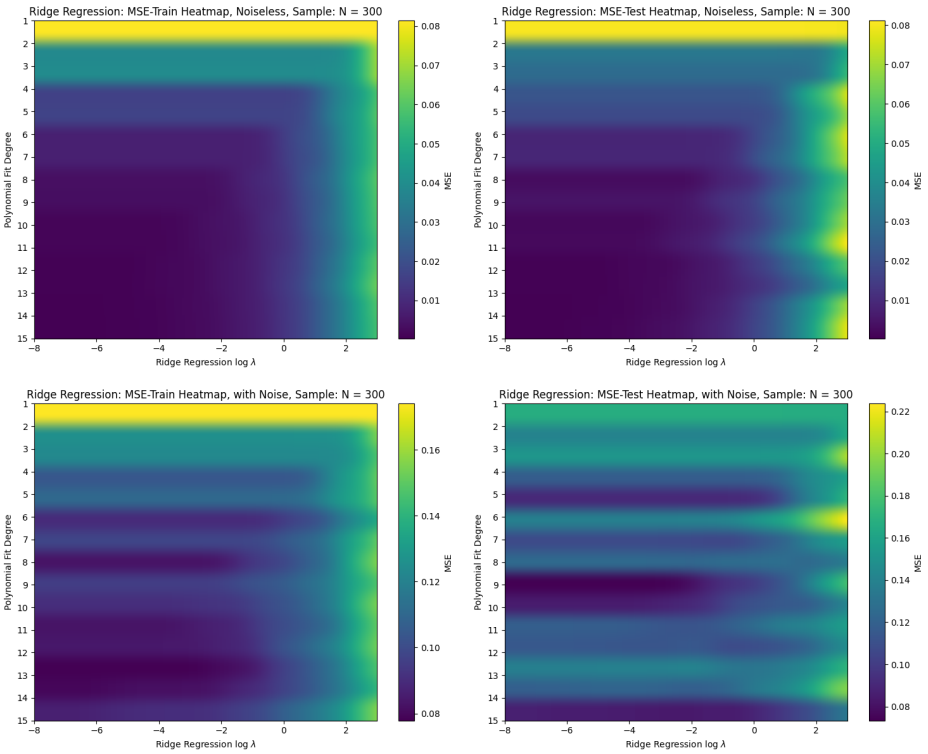

Figure 6: The Heatmap shows MSE of Ridge regression as a function of the regularization parameter λ and model complexity P. Noise: $N(0, \sigma = 0.3)$. When $\lambda \sim \mathcal{O}(X_{ij}^2)$ the model optimization is dictated by the penalty term, effectively fitting with a constant vector. The non-monotonic colour gradient along P implies an interpretation that regularization decreases the risk of overfitting at larger complexities.

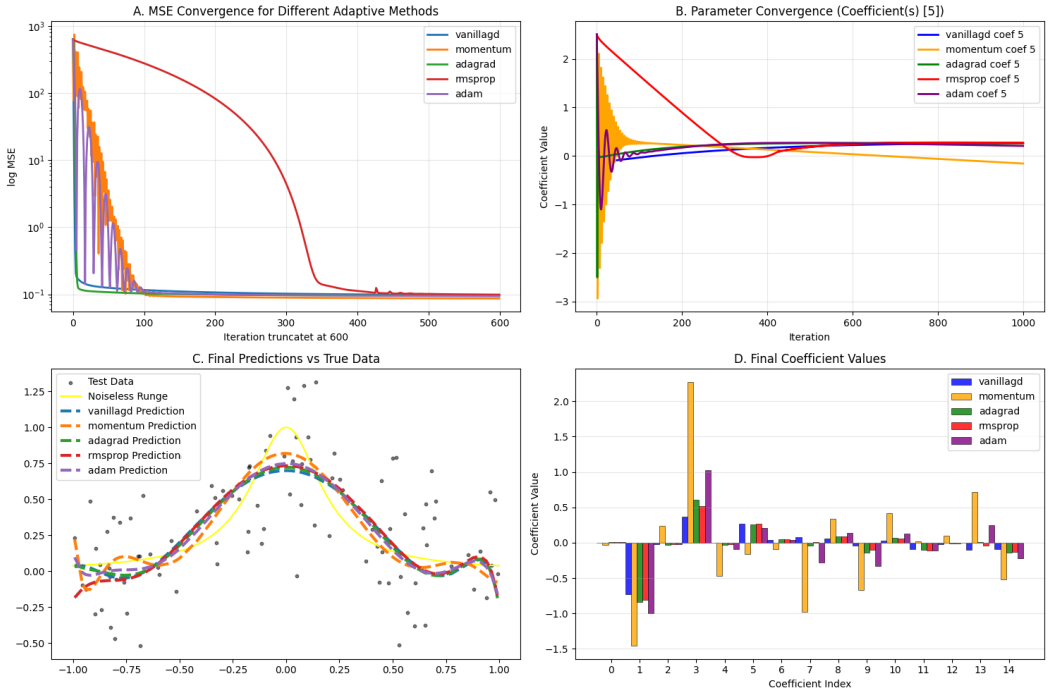

Figure 7: Test D: Adaptive Learning Rate OLS GD. Momentum (orange) predicts best near the origin but suffers oscillations near the edges. AdaGrad (green) converges fast in convex regions but fails to improve edge fluctuations due to cumulative learning-rate scaling. RMSProp (red) converges slowest yet follows the fit, diverging only near the boundary. Adam (purple) combines momentum and RMS moments with bias correction, yielding the most balanced fit across flat and curved regions.

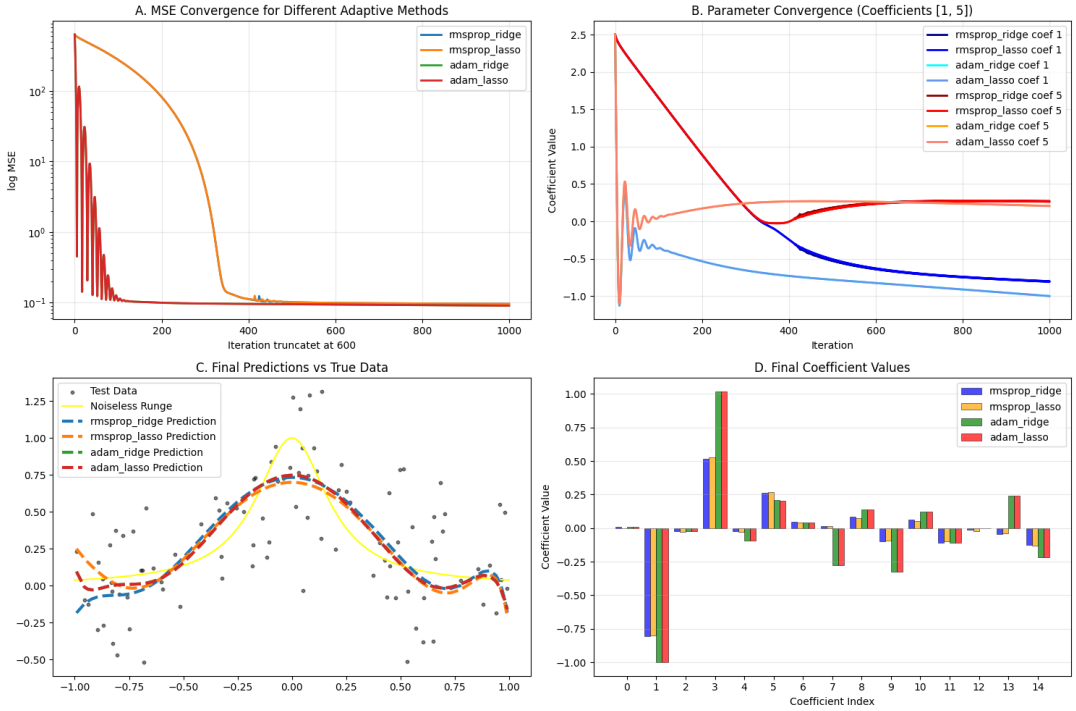

Figure 8: Test E: Lasso and Ridge RMS-ADAM GD. Globally the performance has not improved on the OLS adaptive GD. For RMSProp, the choice of $L1$ (Lasso) opposed to $L2$ (Ridge) regularization affects the fit near the edges. Ridge regression displays a more stable performance in said regime.

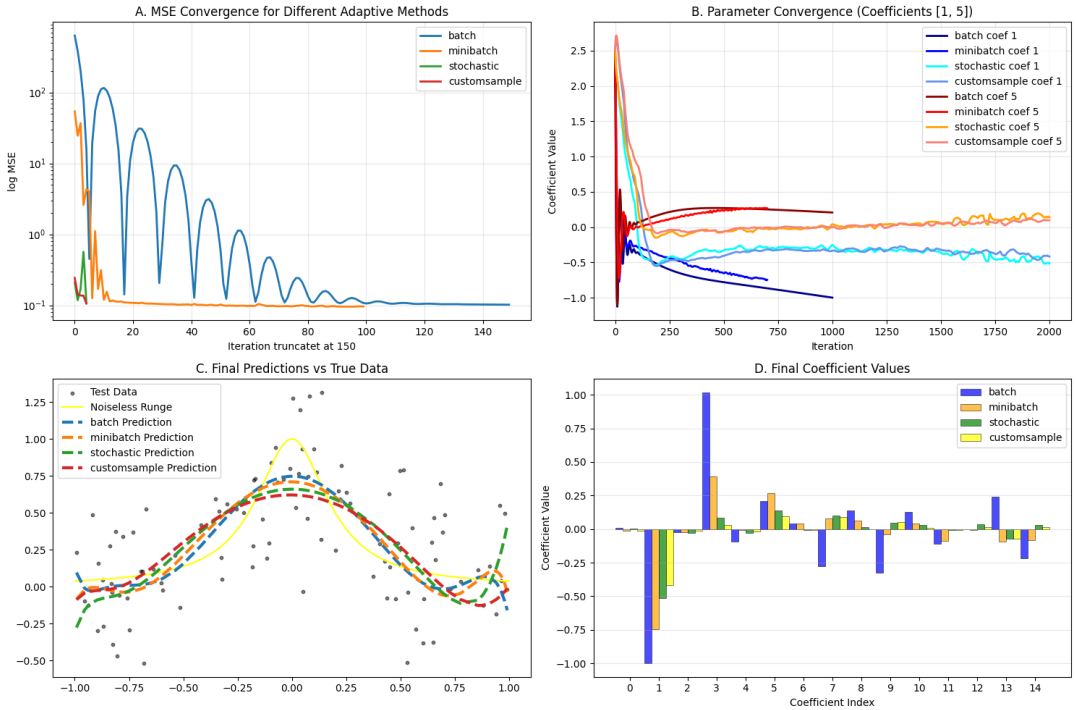

Figure 9: Test F: Batch, Mini-Batch and Stochastic GD modes. Stochastic methods have accelerated convergence compared to batch, but they find very different coefficients. The importance sampling modification improves the edge fit, but worsens the overall prediction by broadening the bell shape. Mini-batch method with batch size $S = 64$, is closets to the batch GD fit.

Introducing adequate regularization stabilized the coefficient values and in particular smoothed out the variance burst, deferring the risk of overfitting to higher complexities. For more noisy samples, this benefit of regularization diminished. Still, Ridge regression proves to be more robust than OLS in case of sparse data samples. Indirectly, is was shown that Bootstrapping is a working option to obtain more test samples simply through resampling from the same test distribution.

For slope tracing, analytical OLS and Ridge give a good prediction closer to the center but add oscillations near the edges. Gradient Descend is a more robust optimization method, particularly for non-linear cases where said analytic expressions don't exist. GD fit broadens the bell shape, somewhat collapsing it near the origin.

Adaptive learning rate proves instrumental for non-trivial shapes, like the Runge function. We can remark RMSProp and ADAM as particularly potent methods. These methods combine moments that work best for convex slopes and flat regions. This allows the method to adapt to different topologies in parameter space and not become dependent on *one* learning rate scaling. ADAM achieves an improved fit around the center and traces the Runge slope without diverging or oscillating up to the edges.

Finally, stochastic and mini-batch GD demonstrated accelerated convergence, but at the cost of noisier updates resulting in a worse overall fit. The bell shape broadens, degrading specifically near the origin, though the edge behavior is as impressive. Notably, SDG with Importance Sapling expresses this tradeoff the most. Mini-batching approximated batch GD the most, as a faster yet more noisy variant. In general, for larger problems where convergence is essential or if only certain regions are important to fit precisely, mini-batching and SDG have advantages to consider.

[1] A. Peirce, *Runge phenomenon and piecewise polynomial interpolation (lecture 3)* (2017), accessed October 5, 2025, URL `https://personal.math.ubc.ca/~peirce/M406_Lecture_3_Runge_Phenomenon_Piecewise_Polynomial_Interpolation.pdf`.

[2] A. Kværnø and M. Grasmair, *Interpolation* (2022), accessed October 5, 2025, URL `https://www.math.ntnu.no/emner/TMA4130/2022h/html_notes/Interpolation.html`.

[3] A. Kværnø, *TMA4215 Numerical Mathematics: Lecture notes* (Department of Physics, Department of Mathematics, NTNU, Norway, 2016), page: 5-7, URL `https://www.math.ntnu.no/emner/TMA4215/2016h/notater/collection_of_notes.pdf`.

[4] M. Hjorth-Jensen, *Computational Physics Lecture Notes 2015* (Department of Physics, University of Oslo, Norway, 2015), week: 35, 36, 37, 38, 39, URL `https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html`.

[5] T. Hastie, R.Tibshirani, and J.Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. Springer Series in Statistics* (Springer, New York, 2009), URL `https://link.springer.com/book/10.1007%2F978-0-387-84858-7`.

[6] I. Goodfellow, Y. Bengio, and A. Courville, in *Deep Learning* (MIT Press, 2016), accessed October 5, 2025, URL `https://www.deeplearningbook.org/contents/optimization.html`.

[7] H. J. C. Berendsen and W. F. van Gunsteren, in *Molecular Dynamics Simulation of Statistical-Mechanical Systems: Proceedings of the International School of Physics "Enrico Fermi"*, edited by G. Ciccotti and W. G. Hoover (North-Holland, Amsterdam, 1986), pp. 56–60, course XCVII, Varenna on Lake Como, 23 July–2 August 1985, URL `http://inis.jinr.ru/sl/Simulation/Ciccotti,Hoover_ed,_MD_Simulation_of_Stat_Mech_Systems,1986.pdf`.